# L9. AVANTAJELE ȘI DEZAVANTAJELE PROGRAMĂRII ÎN LIMBAJ DE ASAMBLARE. APLICAȚII PENTRU CITIREA COMUTATOARELOR ȘI AFIȘAREA PE LED-URI.

## 1. Obiective

Prin parcurgerea acestei ședințe de laborator studenții vor fi capabili:

- Să înveţe să utilizeze instrucţiunile de iniţializare a registrelor, de transfer a datelor între registre, de incrementare şi decrementare a registrelor, precum şi alte operaţii aritmetice şi logice;
- Să învețe să declare și să definească constante și variabile simple sau șiruri amplasate în memorie sau să rezerve spațiu static de memorie pentru variabilele programului;
- Să înveţe să utilizeze instrucţiunile de I/E pentru citirea porturilor de intrare şi înscrierea porturilor de ieşire;
- Să învețe să utilizeze instrucțiunea de selecție și iterația cu număr cunoscut de pași;
- Să înveţe să efectueze operaţii aritmetice şi logice direct cu variabile amplasate în memorie.

## 2. Sintaxa asamblorului ASM85

Pentru a scrie fişiere sursă pentru acest asamblor nu este suficient să cunoaştem instrucţiunile microprocesorului 8085, ci şi formatul şi sintaxa fişierului sursă. Astfel, un fişier sursă în limbaj de asamblare pentru ASM85 poate conţine una sau mai multe linii sursă. O linie sursă are următorul format:

**<etichetă>[:] <instrucţiune> <operanzi> [;]<comentariu>**

- Se pot folosi în orice câmp atât litere mici cât şi majuscule, în orice combinaţie (asamblorul nu face diferenţa dintre litere mici şi litere mari);
- Eticheta este opţională, fiind un nume simbolic dat adresei la care se va stoca instrucţiunea următoare.
- În cazul în care există, eticheta trebuie să înceapă din prima coloană a liniei sursă şi trebuie separată de câmpul următor cel puţin printr-un spaţiu sau un TAB. Opţional, eticheta poate fi urmată de un caracter **:**. Pentru compatibilitate cu alte asambloare, se va folosi întotdeauna caracterul **:** după numele etichetei.
- Câmpul instrucţiune poate fi amplasat pe aceeaşi linie cu eticheta sau într-o linie următoare.
- Dacă nu există o etichetă, atunci instrucţiunea nu trebuie să înceapă din coloana 1, ci trebuie precedată de cel puţin un spaţiu sau un TAB.
- Dacă instrucţiunea are operanzi, aceştia trebuie separaţi de numele instrucţiunii prin cel puţin un spaţiu sau un TAB.

- Dacă instrucţiunea are doi operanzi, atunci aceştia vor fi separaţi printr-o virgulă.
- Câmpul opţional de comentariu trebuie separat de operanzi prin cel puţin un spaţiu sau un TAB şi poate fi precedat de caracterul ;. Pentru compatibilitate cu alte asambloare, se va folosi întotdeauna ; înainte de comentariu.

# 3. Instrucțiuni în limbaj de asamblare

## 3.1. Directive de asamblare (pseudo-instrucțiuni) pentru definirea variabilelor

Forma generală a celor 4 directive este:

| | |
|---|---|
| **DB <expr1>[,<expr2>,<expr3>,...]** | Define Byte |
| **DW <expr1>[,<expr2>,<expr3>,...]** | Define Word |
| **DS <expression>** | Define Storage |
| **STR 'text string'** | String |

| Directiva | |
|---|---|
| **Define Byte (DB)** | **[etichetă:] DB listă**<br>Definește valoarea **etichetă** ca fiind adresa de început a unei zone de memorie alocată static, organizată pe octeți și inițializată cu valorile numerice ale elementelor specificate în **listă**. Elementele specificate în listă pot fi valori numerice, constante simbolice, șiruri de caractere ASCII între apostroafe sau expresii aritmetice și logice. DB este folosită de obicei pentru a defini date de 1 octet, constante sau variabile inițializate static. |
| **Define Word (DW)** | **[etichetă:] DW listă**<br>Directiva DW este similară cu DB, cu specificația că zona de memorie alocată este organizată pe cuvinte de 2 octeți. Lista poate fi formată din valori numerice, etichete de instrucțiuni sau de date |
| **Define Storage (DS)** | **[etichetă:] DS expresie**<br>Directiva DS definește valoarea **etichetă** ca fiind adresa de început a unei zone de memorie neinițializată, alocată static și având dimensiunea **expresie**. Câmpul expresie poate să conțină fie un număr, fie o constantă simbolică sau o expresie aritmetică. |

### 3.1.1. Definirea constantelor simbolice

Atunci când o constantă are o semnificație bine definită, pentru o mai bună lizibilitate a programului se preferă atribuirea unui nume simbolic acelei constante și utilizarea numelui în

cadrul instrucțiunilor în locul valorii. Pentru definirea constantelor simbolice, ASM85 oferă directiva de asamblare (pseudo-instrucțiunea) EQU (EQUate).

Constantele numerice încep întotdeauna cu o cifră:

| Codificare | Baza | Exemplu |
|---|---|---|
| **Zecimal** | 10 | 179 |
| **Binar** | 2 | 11110011b |
| **Octal** | 8 | 731Q |
| **Hexazecimal** | 16 | 0E8h |
| **ASCII** | - | # în față |

**const EQU expresie** $const \leftarrow expresie$
**PIN EQU 41h** $PIN \leftarrow 41h$
**POUT EQU 40h** $POUT \leftarrow 40h$

### 3.1.2. Definirea constantelor și a variabilelor

**const:** **DB 'A'** La adresa const se definește un caracter A.
**tabconst:** **DB 10,11,12,13,50h,0** La adresa tabconst se definește un tablou de octeți inițializat cu valorile din listă.
**const:** **DW 0** La adresa const de definește o constantă de 16 biți cu valoarea 0.
**tab:** **DW 1000, 253, 12Ah, 0FFFFh, 1001000010110111b** La adresa tab se definește un tabel de constante de 16 biți cu valorile din listă.
**buffer:** **DS 32** La adresa buffer începe un tabel de 32 de octeți.
**ptr:** **DS 2** La adresa ptr începe o variabilă de 2 octeți.
**contor:** **DS 1** La adresa contor se află o variabilă de 1 octet.
**tabel:** **DS 2*nr_elem** La adresa tabel începe un tablou cu elemente de 16 biți.
**mesaj:** **STR "Meniu comenzi"** La adresa mesaj se definește un șir ASCII cu conținutul Meniu comenzi.

## 3.2. Instrucțiuni de intrare – ieșire

În cazul microprocesorului 8085, adresarea porturilor (I/O) se face pe 8 biți, deci adresa unui port poate să fie cuprinsă între 00h și FFh. Transferul I/O se realizează prin intermediul registrului A astfel: starea unui port de intrare (I) poate fi încărcată numai în A, iar comanda unui port de ieșire (O) se poate face numai cu octetul din A. Vorbim în acest caz de adresarea implicită.

**IN *port*** Input from port $A \leftarrow (port)$
$port = 00h \div FFh$

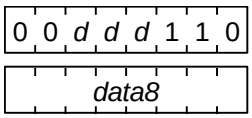
**OUT *port*** Output to port $(port) \leftarrow A$
$port = 00h \div FFh$

### 3.3. Inițializarea registrelor

#### 3.3.1. Inițializarea registrelor pe 8 biți

În această categorie intră registrele A, B, C, D, E, H şi L. Formatul general al unei astfel de instrucţiuni este:

**MVI** ***r, data8*** MoVe Immediate data data8 to register r $r \leftarrow data8$




În limbaj de asamblare, instrucţiunea are numele (mnemonica) *MVI* şi doi operanzi: numele registrului care se iniţializează (*r*) şi valoarea care se încarcă în registrul respectiv (*data8*). Primul octet reprezintă codul operației și cuprinde o serie de biți fixați care reprezintă codul operației și trei biți **(ddd**) care codifică registrul folosit de instrucțiune (registrul destinație) astfel:

| **r** | **B** | **C** | **D** | **E** | **H** | **L** | | **A** |
|---|---|---|---|---|---|---|---|---|
| **ddd** | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

Cel de-al doilea octet al instrucțiunii este reprezentat de o valoare numerică pe 8 biți care este încărcată în registrul specificat de opcod.

În limbajul de asamblare ASM85, un astfel de operand poate fi exprimat printr-o constantă numerică, o constantă simbolică a cărei valoare a fost definită în cadrul fișierului sursă sau printr-o expresie aritmetică sau logică cu constante numerice și/sau simbolice.

#### 3.3.2. Inițializarea registrelor pe 16 biți

Când se dorește inițializarea unui registru pereche cu o anumită valoare pe 16 biți, pot fi utilizate două instrucțiuni MVI corespunzătoare celor doi octeți. O soluție mai eficientă constă în utilizarea instucțiunii LXI care permite încărcarea simultană a registrelor pereche cu o valoare pe 16 biți.

**LXI** ***rp, data16*** Load register pair Immediate rp$\leftarrow$ *data16*

În limbaj de asamblare, instrucțiunea LXI are doi operanzi: rp - registrul pereche care este inițializat și data16 – valoarea care se încarcă în registrul pereche respectiv. Litera X din numele instrcțiunii precizează că registrul desemnat de primul operand este un registru pereche.

Dacă pentru inițializarea registrelor pereche B, D și H instrucțiunea LXI poate fi înlocuită cu două instrucțiuni MVI, pentru inițializarea registrului SP acest lucru nu este posibil, deoarece SP este accesibil numai ca registru pe 16 biți.

### 3.3.3. Accesul direct la locațiile de memorie

În cazul în care la momentul scrierii programului se cunoaște adresa la care este amplasată o variabilă (ca valoare numerică sau ca nume simbolic), se pot folosi pentru acces instrucțiuni cu adresare directă. Acestea transferă unul sau doi octeți între memorie și anumite registre interne ale microprocesorului.

**LDA addr** Load Accumulator Direct $A \leftarrow addr$

**STA addr** Store Accumulator Direct $addr \leftarrow A$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| LOW(addr) | | | | | | | |
| HIGH(addr) | | | | | | | |

**LHLD addr** Load H and L Direct $L \leftarrow addr$, $H \leftarrow addr + 1$

**SHLD addr** Store H and L Direct $addr \leftarrow L$, $addr + 1 \leftarrow H$

Transferul se poate realiza numai prin intermediul acumulatorului. Nu există instrucțiuni care să realizeze transfer direct între două locații de memorie. Transferul între două locații de memorie se poate face în doi pași: citirea din locația sursă și înscrierea în locația destinație prin intermediul acumulatorului.

Instrucțiunile care transferă doi octeți (LHLD și SHLD) efectuează transferul între registrul pereche H (registrele H și L) și două locații de memorie (de la adresa addr și addr+1). Transferul se poate realiza numai prin intermediul registrului pereche H. Nu există instrucțiuni care să efectueze transferul direct al unui cuvânt de 16 biți între două variabile pe 16 biți din memorie. Tranferul se realizează similar cu transferul pentru date pe 8 biți prezentat anterior.

## 3.4. Transferul datelor între registrele interne

Formatul general al acestor instrucţiuni este:

**MOV** ***r1,r2*** MOVe register to register $r1 \leftarrow r2$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | *d* | *d* | *d* | *s* | *s* | *s* |

Primul operand este registrul destinaţie (*r1*), cel de-al doilea este registrul sursă (*r2*). Conţinutul registrului sursă se copie în registrul destinaţie, astfel încât după execuţia instrucţiunii, registrul *r1* conţine o copie a datei din registrul *r2*. Conţinutul anterior al registrului *r1* se pierde. Registrele *r1* şi *r2* pot fi oricare dintre cele 7 registre interne de 8 biţi.

### 3.5. Incrementarea și decrementarea registrelor pe 8 biți

Indicatorii de condiţii sunt afectaţi de instrucţiunile care execută operaţii aritmetice şi logice. Pentru început, să considerăm cele mai simple operaţii aritmetice: incrementarea şi decrementarea registrelor interne.

Formatul general al acestor instrucţiuni este:

| | | | | |
|---|---|---|---|---|
| **INR** | *r* | INcRement register | *r* ← *r*+1 | 0 0 *d d d* 1 0 0 |
| **DCR** | *r* | DeCRement register | *r* ← *r* -1 | 0 0 *d d d* 1 0 1 |

**Observaţii:**

- Indicatorii sunt afectaţi numai de rezultatul unor operaţii aritmetice şi logice, nu de conţinutul static al unor registre.
- Registrele pot fi modificate prin instrucţiuni de transfer fără a afecta indicatorii de condiţii din acel moment.
- Instrucţiunile de incrementare/decrementare pe 8 biţi afectează toţi indicatorii conform unei operaţii de adunare/scădere cu 1, **cu excepţia indicatorului CY**.

### 3.6. Instrucțiuni aritmetice și logice

#### 3.6.1. Instrucțiunile aritmetice

**Instrucțiunile aritmetice** care utilizează registrele interne de 8 biți au următorul format general:

| | | |
|---|---|---|
| **ADD r** | ADD register r to accumulator | $A \leftarrow A + \mathrm{r}$ |
| **ADI data8** | Add Immediate data to accumulator | $A \leftarrow A + data8$ |
| **SUB r** | SUBstract register from accumulator | $A \leftarrow A - r$ |
| **SUI data8** | Substract Immediate data from accumulator | $A \leftarrow A - data8$ |

Cele două operații aritmetice de bază sunt adunarea și scăderea. Registrul acumulator A este folosit implicit ca prim operand. Cel de-al doilea operand poate fi o constantă pe 8 biți sau unul din cele 7 registre pe 8 biți inclusiv acumulatorul.

Operațiile se realizează în aritmetica numerelor întregi fără semn, reprezentate pe 8 biți. Scăderea se realizează prin adunarea primului operand cu cel de-al doilea operand reprezentat în cod complementar. Rezultatul este reținut întotdeauna în registrul acumulator (primul operand se pierde).

### 3.6.2. Instrucțiuni logice

Instrucțiunile logice care utilizează registrele interne de 8 biți au următorul format general:

| | | |
|---|---|---|
| **ANA r** | And register r with A | $A \leftarrow A \wedge \mathrm{r}$ |
| **ANI data8** | And Immediate data8 with A | $A \leftarrow A \wedge \mathrm{data8}$ |
| **ORA r** | OR register r with A | $A \leftarrow A \vee \mathrm{r}$ |
| **ORI data8** | OR Immediate data8 with A | $A \leftarrow A \vee \mathrm{data8}$ |
| **XRA r** | eXclusive oR register r with A | $A \leftarrow A \oplus \mathrm{r}$ |
| **XRI data8** | eXclusive oR Immediate data8 with A | $A \leftarrow A \oplus \mathrm{data8}$ |
| **CMA** | CoMplement A | $A \leftarrow \bar{A}$ |

Cele trei operații de bază sunt ȘI, SAU și XOR (SAU EXCLUSIV). Ele utilizează implicit registrul acumulator ca prim operand. Cel de-al doilea operand poate fi o constantă pe 8 biți sau unul din cele 7 registre interne pe 8 biți, inclusiv acumulatorul. Operațiile se realizează bit cu bit pe ranguri. Rezultatul este reținut mereu în registrul acumulator și primul operand se pierde.

Aceste instrucțiuni afectează toți indicatorii de condiții (CY și AC sunt forțați în 0).

### 3.6.3. Instrucțiuni de rotație

O categorie specială a instrucțiunilor logice este cea a instrucțiunilor de rotire sau deplasare. Operandul este implicit conținutul acumulatorului, care poate fi deplasat spre stânga sau spre dreapta cu o poziție. Bitul care iese printr-o extremitate este încărcat automat în indicatorul CY. Pe la cealaltă extremitate a acumulatorului se introduce fie același bit care iese din acumulator, fie bitul aflat anterior in CY.

| | | |
|---|---|---|
| **RLC** | Rotate A Left with Carry | $A_{n+1} \leftarrow A_n, n = 0 \div 6$<br>$CY \leftarrow A_7$<br>$A_0 \leftarrow A_7$ |
| **RRC** | Rotate A Right with Carry | $A_n \leftarrow A_{n+1}, n = 0 \div 6$<br>$CY \leftarrow A_0$<br>$A_7 \leftarrow A_0$ |
| **RAL** | Rotate A Left through Carry | $A_0 \leftarrow CY$<br>$CY \leftarrow A_7$<br>$A_{n+1} \leftarrow A_n \; n = 0 \div 6$ |
| **RAR** | Rotate A Right through Carry | $A_7 \leftarrow CY$<br>$CY \leftarrow A_0$<br>$A_{n+1} \leftarrow A_n \; n = 0 \div 6$ |

Aceste instrucțiuni afectează doar indicatorul CY.

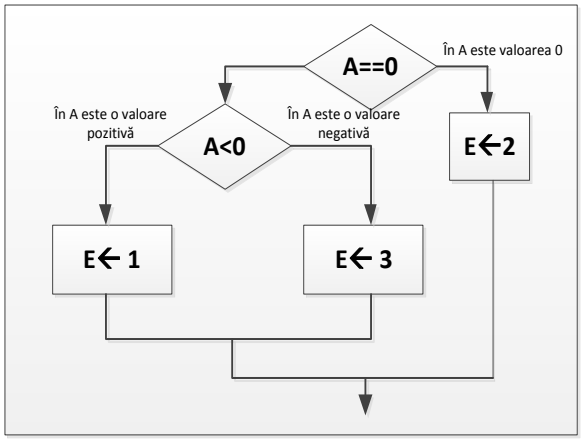
### 3.6.4. Instrucțiuni de comparare

Instrucțiunile de comparare sunt considerate instrucțiuni hibride, deoarece deși operația efectuată este una aritmetică (scăderea), rezultatul aritmetic nu este reținut, ci sunt poziționați indicatorii de condiții ca la o scădere obișnuită.

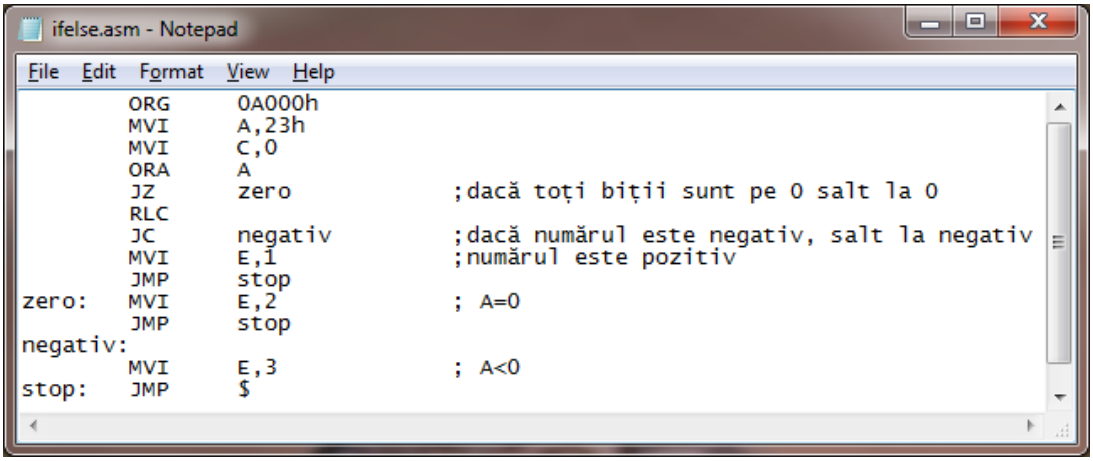
| | | |
|---|---|---|
| **CMP r** | CoMPare register with accumulator | $A - r$ |
| **CPI data8** | ComPare Immediate data with accumulator | $A - data8$ |

Folosind instrucțiuni logice se poate izola un anumit bit sau o anumită combinație de biți dintr-un octet, astfel încât valoarea bitului sau a combinației de biți să afecteze indicatorii de condiții. Starea acestora va putea fi testată direct sau cu ajutorul unor instrucțiuni de comparare pentru a lua decizii de continuare a programului pe anumite ramuri folosind instrucțiunea de salt condiționat.

### 3.7. Instrucțiunea de selecție

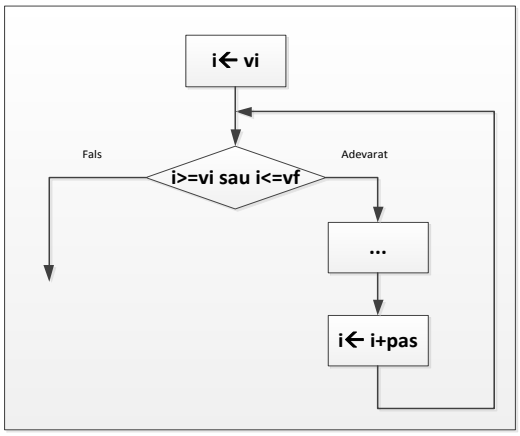



Dacă A==0
    E <-- 2
Altfel
    Dacă A<0
        E <-- 3
    Altfel
        E <-- 1
    Sf. dacă
Sf. dacă

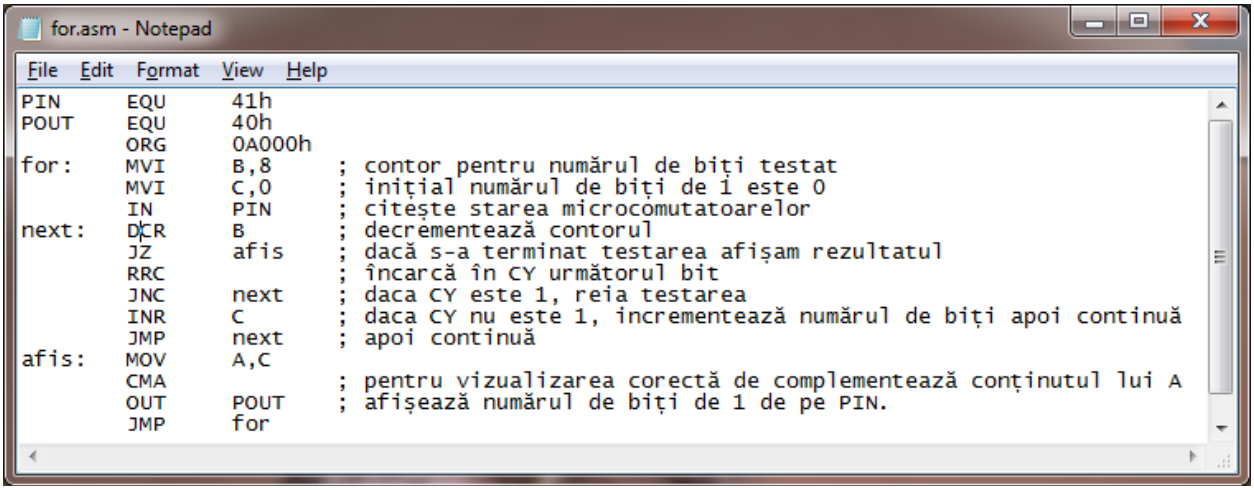


Programul de mai sus testează valoarea din acumulator și scrie în registrul E o valoare astfel: 2 - în cazul în care valoarea din acumulator este 0; 1 - în cazul în care în registrul acumulator avem o valoare pozitivă și 3 - în cazul în care valoarea din acumulator este negativă.

### 3.8. Instrucțiunea de iterație cu număr cunoscut de pași




Pentru i<--vi; i<=vf sau i>=vi; i <-- i+pas
Secvență instrucțiuni
Sf. pentru




Programul descris mai sus, numără biții de pe portul de intrare conectat la microcomutatoare și afișează valoarea rezultată pe ledurile conectate la portul de ieșire.

### 3.9. Instrucțiuni cu adresare indirectă

Atunci când nu se cunoaște adresa la care este amplasată o variabilă, adresa fiind calculată la momentul execuției programului, se vor folosi instrucțiunile cu adresare indirectă. În instrucțiunile prezentate anterior, se face referire la cele 7 registre interne de 8 biți (A,B,C,D,E,H,L) codificare pe 3 biți ddd. Combinația 110 nu era încă alocată, va indica un operand din memorie de la adresa conținută în registrul pereche H. În limbaj de asamblare un astfel de operand va fi indicat prin litera M.

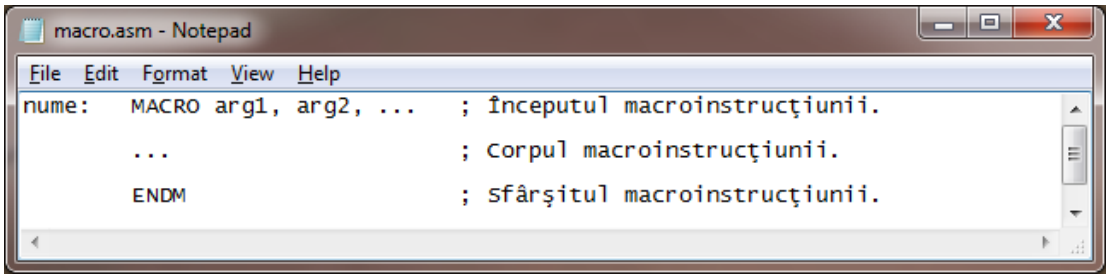
| r | B | C | D | E | H | L | M | A |
|---|---|---|---|---|---|---|---|---|
| ddd | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

Instrucțiunile care admit operanzi amplasați în memorie sunt prezentate în cele ce urmează:

| | | |
|---|---|---|
| **MVI M, data8** | MoVe Immediate data to Memory | $HL \leftarrow data8$ |
| **MOV M,r** | MOVe register to Memory | $HL \leftarrow r$ |
| **MOV r,M** | MOVe Memory to register | $r \leftarrow HL$ |
| **INX rp** | INcrement register pair | $rp \leftarrow rp + 1$ |
| **DCX rp** | DeCrement register pair | $rp \leftarrow rp - 1$ |
| **INR M** | INcRement Memory | $HL \leftarrow HL + 1$ |
| **DCR M** | DeCRement Memory | $HL \leftarrow HL - 1$ |
| **ADD M** | ADD Memory to accumulator | $A \leftarrow A + HL$ |
| **SUB M** | SUBstarct Memory from accumulator | $A \leftarrow A - HL$ |
| **ANA M** | And Memory with A | $A \leftarrow A \wedge HL$ |
| **ORA M** | OR Memory with A | $A \leftarrow A \vee HL$ |
| **XRA M** | eXclusive oR Memory with A | $A \leftarrow A \oplus HL$ |
| **CMP M** | CoMPare Memory with accumulator | $\mathrm{A - HL}$ |

### 3.10. Macroinstrucțiuni

Programarea folosind macroinstrucțiuni este o tehnică de optimizare a codului sursă prin reducerea dimensiunii și modularizare. Astfel, secvenţe identice care apar de mai multe ori într-un program, eventual cu alţi parametri pot fi înlocuite printr-o macroinstrucțiune. O macroinstrucțiune poate fi definită după cum urmează:



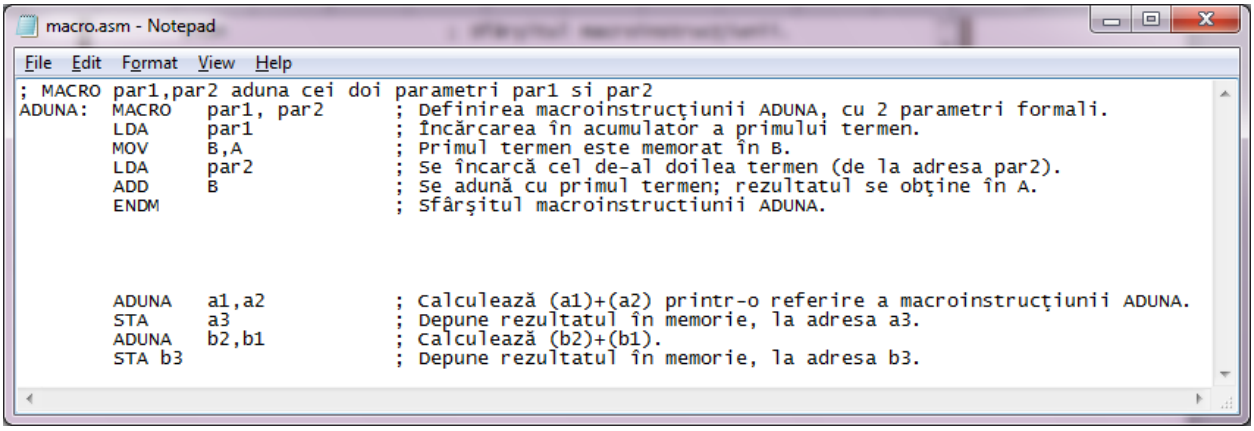

După definirea sa, macroinstrucțiunea poate fi apelată ori de câte ori este nevoie utilizând numele său, urmat de parametrii dorițI.

**Exemplu:**

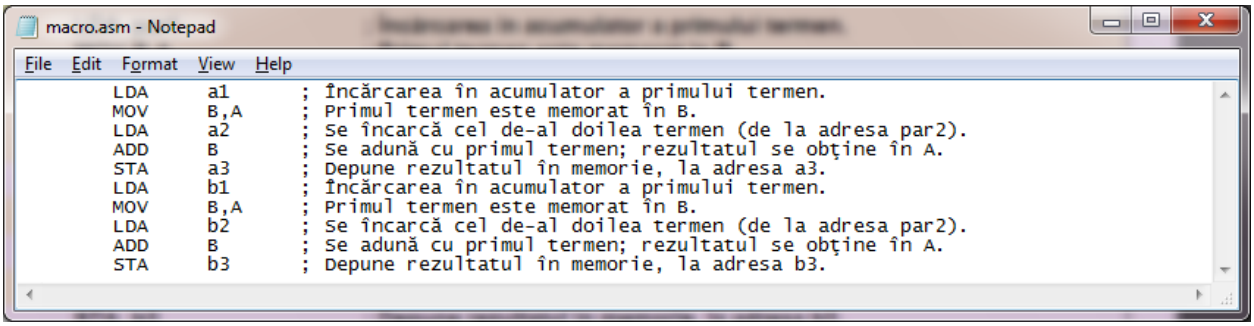
Să se scrie o macroinstrucţiune pentru adunarea a două locaţii de memorie de 1 octet cu stocarea rezultatului în registrul acumulator. Folosind această macroinstrucţiune:

- să se adune conţinutul locaţiei *a1* cu octetul de la adresa *a2*, rezultatul să se depună la *a3*

- să se adune octeţii de la adresele *b1* şi *b2*, iar rezultatul să se memoreze la adresa *b3*.




La asamblare, macroasamblorul va expanda toate referirile la macroinstrucţiune prin înlocuirea liniei de referire cu corpul macroinstrucţiunii și a argumentelor formale cu parametrii efectivi după cum urmează:




## 4. Aplicații propuse

**4.1.** Să se scrie codul necesar pentru definirea unei constante simbolice:

a) Pentru portul de intrare;

b) Pentru portul de ieșire.

**4.2.** Să se scrie codul necesar pentru definirea:

a) caracterului **C** la adresa **const**;

b) tabloului de octeți inițializat cu **34,45,30h** la adresa **tabconst**;

c) o constantă inițializată cu **8000h** la adresa **const**;

d) un șir ASCII cu conținutul **HELLO WORLD** la adresa **mesaj**;

e) un tablou de **16 octeți** la adresa **buff**;

f) o variabilă pe **2 octeți** la adresa **var**;

g) un tabel cu elemente pe **doi octeți** la adresa **tab**.

**4.3.** Să se testeze valoarea din registrul acumulator și scrie în registrul E valoarea
- 2 în cazul în care valoarea din registrul acumulator este 0;
- 1 în cazul în care în registrul acumulator avem o valoare pozitivă și
- 3 în cazul în care valoarea din registrul acumulator este negativă.

Să se explice instrucțiune cu instrucțiune codul programului.

**4.4.** Să se numere biții de 1 din portul de intrare conectat la microcomutatoare, apoi valoarea rezultată să se afișeze pe ledurile de pe portul de ieșire. Să se identifice instrucțiunea **for/pentru**. Să se explice instrucțiune cu instrucțiune codul programului.

**4.5.** *Să se scrie înscrie în registrul A valoarea C3h și să se aplice măștile M1 = FFh, M2 = 00h, M3 = 0Fh și M4 = F0h:

a) SAU M1; c) SAU M3; e) AND M1; g) AND M3; i) XOR M1;
b) SAU M2; d) SAU M4; f) AND M2; h) AND M4; j) XOR M2.

A) Se consideră A inițializat cu valoarea C3h de fiecare dată când se aplică masca;
B) Se consideră A inițializat la început cu valoarea C3h.

Să se precizeze codul programului respectiv și rezultatele obținute în cele două cazuri.

**4.6.** **Cât timp microcomutatorul 5 este ON, să se deplaseze continuu un led aprins spre dreapta, iar atunci când devine OFF, deplasarea să se oprească.
Indicații: vezi Lucrarea nr. 8 – problema 5.3

**4.7.** **Să se scrie codul programului care deplasează continuu două leduri aprinse, primul către dreapta, al doilea către stânga.

## 5. Referințe bibliografice

[1] C.Huțanu, M.Postolache, *Sisteme cu microprocesoare*, Editura Academică, Iași, 2001.
[2] Gh.Toacșe, *Introducere în microprocesoare*, Editura Științifică și Enciclopedică, București, 1985.
[3] ***, Universal Trainer, Lab Manual for Board Revisions R1 and R2, EMAC INC, 1993.
[4] ***, Universal Trainer, Reference Manual, EMAC INC, 1993.
[5] ***, Universal Trainer, Self Instruction Manual, EMAC INC, 1992.